CSCI 136
Data Structures &
Advanced Programming

Lecture 35
Fall 2018
Announcements

• Final Class 😢

• Help Opportunities 😊
  • Office Hours Next Week: M/T/Th/F: 1-3 pm
  • Review Session: Thursday, Dec. 13: 8-9 pm

• Final Exam is Monday, Dec. 17 😑
  • 9:30-noon in Physics 203
  • Cumulative, but focused on second half of course
  • Sample exam and 2-page study sheet are on-line
Last Time

- Maps & Hashing
Today

• Hashing Wrap-up
• One More Problem
• Course Wrap-up
• SCS Forms
Hashtables: $O(1)$ operations?

- How long does it take to compute a String’s hashCode?
  - $O(s\.length())$

- Given an object’s hash code, how long does it take to find that object?
  - $O(\text{run length})$ or $O(\text{chain length})$ PLUS cost of .equals() method

- Conclusion: for a good hash function (fast, uniformly distributed) and small load factor, we say operations take $O(1)$ time
  - But that’s not strictly true....
## Summary

<table>
<thead>
<tr>
<th></th>
<th>put</th>
<th>get</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted vector</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>unsorted list</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted vector</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>balanced BST</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>array indexed by key</td>
<td>$O(1)^*$</td>
<td>$O(1)^*$</td>
<td>$O(\text{key range})$</td>
</tr>
</tbody>
</table>

*PolitiFact Rating: not quite Pants on Fire*
What Can We Say For Sure?!

For external chaining

• Assuming the hashing function is equally likely to hash to any slot

Theorem: A search will take $O(1 + m/n)$ time, on average

• $n$ is table size, $m$ is number of keys stored
• True for both successful and unsuccessful searches
  • Based on expected chain length
What Can We Say For Sure?!

For open addressing

• Assuming that all probe sequences are equally likely [which is unlikely!]
• Assuming load factor $0 < \alpha < 1$

Theorem: An unsuccessful search will perform, on average, $O(1 + \alpha)$ probes

Theorem: A successful search will perform, on average, $O\left(\frac{1}{\alpha} \log \frac{1}{1-\alpha}\right)$ probes

More probe sequences $\Rightarrow$ better average case
In certain cases, it is possible to design a hashing scheme such that

- Computing the hash takes $O(1)$ time
- There are no collisions
  - Different keys always have different hash values

This is called a *perfect hashing scheme*
Perfect Hashing

If keyspace is smaller than array size

- Handcraft the hashing function
  - Ex: Reserved words in programming languages
- Make array really big
  - Ex: All ASCII strings of length at most 4
    - Hash is 32 bit number
    - Array of size 4.3 billion will suffice
One More Problem!

- Given a graph $G = (V,E)$ where
  - $V = X \cup Y$, with $X \cap Y = \emptyset$
  - Every edge has one vertex in $X$ and one in $Y$
- Find a set of edges $M \subseteq E$ such that
  - No vertex is on more than one edge of $M$
  - $M$ is as large as possible
- $G$ is called a \textit{bipartite graph} and $M$ is called a \textit{maximum matching} of $G$
- Fun facts
  - $G$ is bipartite iff the vertices of $G$ can be 2-colored
  - $G$ is bipartite iff every cycle of $G$ has even length
Finding a Maximum Matching

- Idea: Look for *alternating* path between non-matched vertices
- Use it to *augment* the current matching
- Repeat until you can’t find any more of them.

Amazing Fact
- If $M$ is a matching in a bipartite graph and there is no alternating path the augments $M$, then $M$ is a maximum matching for the graph!

Not too hard to prove
- Uses structure of pairs of matchings
Wrapping Up
## Why Data Structures?

<table>
<thead>
<tr>
<th>Dictionary Structures</th>
<th>put</th>
<th>get</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted vector</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>unsorted list</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted vector</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>balanced BST</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>hash table</td>
<td>$O(1)^*$</td>
<td>$O(1)^*$</td>
<td>$O(\text{key range})$</td>
</tr>
</tbody>
</table>

*On average---with good design---Don’t forget!
Data Structure Selection

• Choice of most appropriate structure depends on a number of factors
  • How much data?
    • Static (array) vs dynamic structure (vector/list)
  • Which operations will be performed most often?
    • Lots of searching? Use an ordered structure
      – If items are comparable!
    • Mostly traversing where order doesn’t matter: List
  • Is worst case performance crucial? Average case?
    • AVL tree vs SplayTree
Why Complexity Analysis?

• Provides *performance* guarantees
  • Captures effects of scaling on time and space requirements

• Independent of hardware or language

• Can guide appropriate data structure selection
Why Correctness Analysis?

- Provides *behavior* guarantees
- Independent of hardware or language
- Reduce wasted effort developing code
- A powerful debugging tool
  - Program incorrect: Try to prove it is correct and see where you get stuck
  - Frequently, such proofs are *inductive*
Why Java?

What makes it worth having to type (or read!)

Map<Airport, ComparableAssociation<Integer, Edge<Airport, Route>>> result = new Table<Airport, ComparableAssociation<Integer, Edge<Airport, Route>>>(());
Why Java?

- Java provides many features to support:
  - Data abstraction: Interfaces
  - Information hiding: public/protected/private
  - Modular design: classes
  - Code reuse: class extension; abstract classes
  - Type safety: types are known at compile-time
- As well as:
  - Parallelism, security, platform independence, creation of large software systems, embeddability in browsers, ...
Why structure(5)?

• Provides a well-designed library of the most widely-used fundamental data structures
  • Focus on core aspects of implementation
    • Avoids interesting but distracting “fine-tuning” code for optimization, backwards compatibility, etc
  • Allows for easy transition to Java’s own Collection classes
• Full access to the source code
  • Don’t like Duane’s HashMap---change it!
Why So Many Labs?

Because it’s fun and you got a chance to

• Implement a (simple) game - Coinstrip
• Learn about textual analysis - WordGen
• Grapple with large search problems
  • Recursion, Two Towers, Exam Scheduling
• Do some data mining - Sorting
• Write (part of) a PL interpreter – PostScript
• Implement Data Structures
  • Linked Lists and Lexicon
• Model and Simulate a Business Process
Want to Learn More?

- **CS 237: Computer Organization**
  - Learn about the many levels of abstraction from high-level language ➔ assembly language ➔ machine language ➔ processor hardware

- **CS 256: Algorithm Design and Analysis**
  - We’ve only scratched the surface of what elegant algorithm and data structure design can accomplish. For a deeper dive, go here.

- A number of CS electives require one of these two courses
Want to Learn More?

• CS 334: Principles of Programming Languages
  • There are many different types of programming languages: imperative, object-oriented, functional, list-based, logic, ... Why!? What is required to support languages of these kinds?

• CS Colloquium
  • Weekly (Fridays at 2:30pm) presentations from active researchers in CS from across the country

• Talk to Faculty and CS Majors
  • They do interesting things!